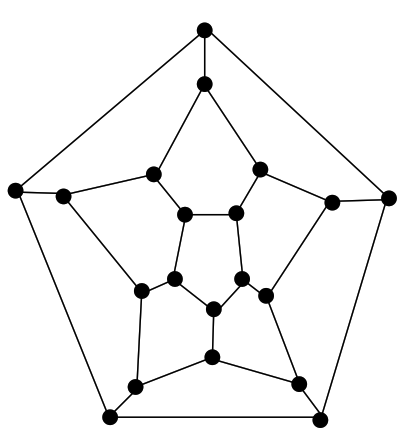
**Lab 12**

**Solam Jung Rana**

1. *Hamiltonian Graphs.* The following graph has a Hamiltonian cycle. Find it.



1. *Vertex Covers.* Create an algorithm for computing the smallest size of a vertex cover for a graph. The input of your algorithm is a set V of vertices along with a set E of edges. Assume you have the following functions available (no need to implement these):
   * computeEndpoints(edge) – returns the vertices that are at the endpoints of the input edge
   * belongsTo(vertex, set) – returns true if the input vertex is a member of the given set

*Hint:* Loop through all subsets of V. For each subset W, check to see if W is a vertex cover. Do this by looping through all edges; for each edge e, check to see if at least one of its endpoints lies in W.

**Algorithm SmallestVertexCoverSize**

**Input: A graph G whose set of vertices is denoted V and set of edges is denoted E**

**Output: A vertex cover U for G of smallest possible size**

**pow ← PowerSet(V)**

**minCover ← V**

**for each U in pow do**

**isCover ← true**

**// verify U is a vertex cover**

**for each e in E do**

**(u,v) ← computeEndpoints(e)**

**if ( !(belongsTo(u, U) and ! belongsTo(v, U))**

**isCover ← false**

**if(isCover and U.size() < minCover.size()) then**

**minCover ← U**

**return minCover.size**

1. The decision problem formulation of the Vertex Cover problem is this: Given a positive integer *k*, and a graph *G*, is there a vertex cover for *G* having size = *k*? Show that this decision problem belongs to *NP.* (The definition of the class NP is given in Lecture 1 (Lesson 1) in (approximately) slide 42.)

**Given a graph G = (V, E) with n vertices**

**Suppose U is a solution to the VertexCover problem**

**We do these steps to verify correctness of U:**

**ii. U ≤ V**

1. **U.size ≤ k**
2. **for each edge e = (e, v) in E, either u is in V or v is in U**

**O(n2)**

1. The goal of this exercise is to devise a feasible algorithm that decides whether an input integer is prime. The key fact that you will make use of is the following:

Fact: There is a function *f*, which runs in O(log *n*) (that is, O(length(*n*))), such that for any odd positive integer *n* and any *a* chosen randomly in [1, *n* - 1], if *f*(*a*, *n*) = 1, then *n* is composite, but if *f*(*a*,*n*) = 0, *n* is “probably” prime, but is in fact composite with probability < ½.

A first try at such an algorithm would be:

*Algorithm* **FirstTry**:

*Input:* A positive integer n

*Ouptut:* TRUE if n is prime, FALSE if n is composite

**if** n % 2 = 0 **return** FALSE

a ← random number in [1, n-1]

**if** f(a,n) = 1

**return** FALSE

**return** TRUE

Notice that **FirstTry** runs in O(log *n*). It also produces a correct result more than half

the time.

What could be done to improve the degree of correctness of **FirstTry** but still

preserve a reasonably good running time? Explain.

**For any k ≥ 1, algorithm returns true 1/2k times**

**For a big value of k then 1/2k is essentially zero.**

**Running algorithm k number of times, provides high probability of prime.**

1. Show that if a graph G has |V| -1 edges and has no cycle, then G is connected.

Hint: Assume G is disconnected, with connected components H1, H2, .. Hk. What can you say about each of these components? Do a computation to show that G must in this case have fewer than |V| - 1 edges (giving a contradiction)

**Each connected component has every vertex linked to all others.**